PL/SQL

Introducción

El lenguaje PL/SQL (Procedural Language Structured Query Language) es una **extensión de programación a SQL** que agrega ciertas construcciones propias de lenguajes estructurados, resultando en un lenguaje más poderoso que SQL.

> Permite gestionar los datos de una base de datos ORACLE introduciendo instrucciones de control de flujo, definición de variables, utilización de procedimientos y funciones, y manejo de excepciones.

La unidad de programación utilizada por PL/SQL es el **bloque**.

- **Bloque Anónimo:** No tiene nombre, reside en el lado del cliente.
- **Bloque con Nombre:** Puede ser un procedimiento, función, o disparador (trigger); se almacena en la BD y está en el servidor.

Estructura del Bloque

La estructura general de un bloque PL/SQL es:

```
DECLARE
  Declaración de variables, cursores, excepciones;
BEGIN
  Instrucciones del bloque
EXCEPTION
  Instrucciones para tratamiento de excepciones
END;
```

- **DECLARE:** Sección opcional para declarar variables, cursores y excepciones. El ámbito de estos elementos es dentro del bloque.
- **BEGIN:** Contiene las instrucciones que se ejecutan dentro del bloque.
- **EXCEPTION:** Sección opcional para el tratamiento de excepciones, generalmente errores en tiempo de ejecución.

Variables

Las variables en PL/SQL se declaran en la sección `DECLARE` o como argumentos de los procedimientos o funciones.

- Los nombres deben empezar con una letra.
- Pueden contener letras, números, $, y #.
- La longitud máxima es de 30 caracteres.
- No distingue mayúsculas de minúsculas (importe, Importe e IMPORTE son la misma variable).

Sintaxis para Declarar una Variable

```
nombrevariable tipodato;
nombrevariable tabla.columna%TYPE;
nombrevariable otravariable%TYPE;
```

Tipos de Datos

- SQL (NUMBER, CHAR, VARCHAR, DATE,...)
- BOOLEAN

También se pueden declarar variables asociándoles el mismo tipo de datos que una columna de una tabla o de otra variable usando `%TYPE`.

Ejemplos

```
importe NUMBER (6);
encontrado BOOLEAN;
cantidad ARTICULO.STOCK%TYPE;
disponible encontrados%TYPE;
```

Asignación de Valores

Se utiliza el operador `:=` para asignar valores a una variable.

```
importe := 1000;
```

También se pueden asignar valores utilizando una consulta `SELECT INTO` que devuelva una única fila.

```
SELECT stock INTO cantidad FROM ARTICULO WHERE codigo = 'AR11';
```

Constantes

```
nombrevariable CONSTANT tipodato := valor;
```

Control de Flujo

Se refiere a las instrucciones básicas para controlar el flujo del programa, incluyendo instrucciones condicionales y bucles.

Ejemplo 1: Bloque PL/SQL Anónimo

Imprimir la fecha de hoy, el stock de los artículos con códigos ART1 y ART2, y la suma de los stocks de ambos artículos.

```
SET SERVEROUTPUT ON
DECLARE
  hoy DATE;
  stock1 ARTICULO.STOCK%TYPE;
  stock2 ARTICULO.STOCK%TYPE;
BEGIN
  hoy := SYSDATE;
  SELECT stock INTO stock1 FROM ARTICULO WHERE codigo = 'AR11';
  SELECT stock INTO stock2 FROM ARTICULO WHERE codigo = 'ART2';
  DBMS_OUTPUT.PUT_LINE (TO_CHAR (hoy, 'DD-MM-YYYY'));
  DBMS_OUTPUT.PUT_LINE ('ART1: ' || stock1);
  DBMS_OUTPUT.PUT_LINE ('ART2: ' || stock2);
  DBMS_OUTPUT.PUT_LINE ('Total articulos:' || (stock1*stock2));
END;
/
```

Instrucciones Utilizadas

Instrucción	Descripción
`SET SERVEROUTPUT`	Variable del sistema que indica si la salida se muestra por pantalla.
`SYSDATE`	Función que devuelve la fecha actual del ordenador.
`DBMS_OUTPUT.PUT_LINE`	Función para imprimir datos. Se encuentra dentro del paquete `DBMS_OUTPUT`.
`TO_CHAR`	Función de conversión de tipos de datos. Convierte el argumento a carácter. También existen `TO_NUMBER` y `TO_DATE`.

`

Instrucción Condicional

Permiten ejecutar diferentes instrucciones dependiendo del valor de una condición. Incluyen estructuras `IF` y `CASE`.

Estructura IF

```
IF condicion THEN
  Instrucciones_si_Verdadero
ELSIF condicion2 THEN
  Instrucciones_si_verdadero_condicion2
ELSE
  Instrucciones
END IF;
```

Ejemplo 2

Imprimir el stock actual del artículo ART1 y un mensaje "Bajo", "Normal" o "Exceso", dependiendo del valor de su stock actual.

```
SET SERVEROUTPUT ON
DECLARE
  stock1 ARTICULO.STOCK%TYPE;
BEGIN
  SELECT stock INTO stock1 FROM ARTICULO WHERE codigo = 'AR11';
  DBMS_OUTPUT.PUT_LINE ('Stock actual: ' || stock1);
  IF stock1 > 5 AND stock1 <= 10 THEN
    DBMS_OUTPUT.PUT_LINE ('Normal');
  ELSIF stock1 < 5 THEN
    DBMS_OUTPUT.PUT_LINE ('Bajo');
  ELSE
    DBMS_OUTPUT.PUT_LINE ('Exceso');
  END IF;
END;
/
```

Estructura CASE

```
CASE expresión
  WHEN valor1 THEN
    Instrucciones
  WHEN valor2 THEN
    Instrucciones
  ELSE
    Instrucciones
END CASE;
```

Bucles

Se utilizan para realizar repetidamente un conjunto de instrucciones hasta que se cumpla una determinada condición.

Estructuras de Bucle

Estructura	Sintaxis
LOOP	`LOOP Instrucciones EXIT WHEN condición; END LOOP;`
WHILE	`WHILE condición LOOP Instrucciones END LOOP;`
FOR	`FOR i IN [REVERSE] v1..v2 LOOP Instrucciones END LOOP;`

Ejemplo 3

Pedir un número por teclado e imprimir la suma de los números desde 1 hasta dicho número, así como las sumas parciales.

```
-- Solución con LOOP
SET SERVEROUTPUT ON
DECLARE
  i NUMBER := 0;
  suma NUMBER := 0;
  v NUMBER;
BEGIN
  v := &numero;
  DBMS_OUTPUT.PUT_LINE ('El valor introducido es: ' || v);
  LOOP
    i := i + 1;
    suma := suma + i;
    DBMS_OUTPUT.PUT_LINE (i || ': ' || suma);
    EXIT WHEN i >= v;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('La suma final es: ' || suma);
END;

-- Solución con WHILE
SET SERVEROUTPUT ON
DECLARE
  i NUMBER := 0;
  suma NUMBER := 0;
  v NUMBER;
BEGIN
  v := &numero;
  DBMS_OUTPUT.PUT_LINE ('El valor introducido es: ' || v);
  WHILE i < v LOOP
    i := i + 1;
    suma := suma + i;
    DBMS_OUTPUT.PUT_LINE (i || ': ' || suma);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('La suma final es: ' || suma);
END;

-- Solución con FOR
SET SERVEROUTPUT ON
DECLARE
  suma NUMBER := 0;
  v NUMBER;
BEGIN
  v := &numero;
  DBMS_OUTPUT.PUT_LINE ('El valor introducido es: ' || v);
  FOR i IN 1..v LOOP
    suma := suma + i;
    DBMS_OUTPUT.PUT_LINE (i || ': ' || suma);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('La suma final es: ' || suma);
END;
```

En los bucles `FOR`, se puede utilizar `REVERSE` para realizar la suma en sentido contrario.

Cursores

Se utilizan para almacenar el resultado de una consulta `SELECT` que generalmente devuelve más de una fila.

Acciones con Cursores

1. **Declarar:**

```
CURSOR nombrecursor IS instrucción_SELECT;
CURSOR nombrecursor (parámetro1 tipodato, parámetro2 tipodato,...) IS in
```

Ejemplos:

```
CURSOR stockbajo IS SELECT codigo, stock FROM ARTICULO WHERE stock < 5;
CURSOR stocknormal IS SELECT codigo, stock FROM ARTICULO WHERE stock >=
```

2. **Abrir:**

```
OPEN nombrecursor;
```

Al abrir un cursor, se ejecuta la consulta `SELECT` asociada y se almacena el resultado en memoria.

3. **Recorrer:**

```
FETCH nombrecursor INTO variable1, variable2,...;
FETCH nombrecursor INTO variablefila;
```

Cada vez que se utiliza `FETCH`, se almacenan los datos de la fila apuntada por el cursor en las variables indicadas en `INTO` y se avanza el puntero a la siguiente fila.

Para declarar la variable fila, se puede usar:

```
nombrevariable nombrecursor%ROWTYPE;
```

Y para acceder a sus campos:

```
nombrevariable.nombre_componente
```

4. **Cerrar:**

```
CLOSE nombrecursor;
```

Cuando ya no se utiliza el cursor, se cierra para liberar la memoria utilizada.

Atributos de Cursores

Atributo	Significado	Valores
%ISOPEN	Indica si el cursor está abierto.	TRUE si está abierto, FALSE en caso contrario.
%NOTFOUND	Indica si el último FETCH no recuperó ninguna fila.	TRUE si no recuperó ninguna fila, FALSE si devolvió una fila, NULL si el cursor está abierto y no se ha realizado ningún FETCH, INVALID_CURSOR si el cursor no está abierto.
%FOUND	Indica si el último FETCH recuperó una fila.	TRUE si recuperó una fila, FALSE si no devolvió una fila, NULL si el cursor está abierto y no se ha realizado ningún FETCH, INVALID_CURSOR si el cursor no está abierto.
%ROWCOUNT	Número total de filas recorridas por el cursor.	Número de filas, INVALID_CURSOR si el cursor no está abierto.

Ejemplo 4: Listado de Artículos con Stock Inferior a 8 Unidades

```
-- Solución versión 1
SET SERVEROUTPUT ON
DECLARE
  CURSOR datos IS
    SELECT codigo, stock
    FROM ARTICULO
    WHERE stock < 8
    ORDER BY stock;
  codigofila ARTICULO.CODIGO%TYPE;
  stockfila ARTICULO.STOCK%TYPE;
  mensaje CHAR(10);
BEGIN
  OPEN datos;
  LOOP
    FETCH datos INTO codigofila, stockfila;
    EXIT WHEN datos%NOTFOUND;
    IF stockfila >= 5 AND stockfila <= 10 THEN
      mensaje := 'Normal';
    ELSIF stockfila < 5 THEN
      mensaje := 'Bajo';
    ELSE
      mensaje := 'Exceso';
    END IF;
    DBMS_OUTPUT.PUT_LINE (codigofila || ' - ' || stockfila || ' - ' || mer
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('Total registros: ' || datos%ROWCOUNT);
  CLOSE datos;
END;

-- Solución versión 2
SET SERVEROUTPUT ON
DECLARE
  CURSOR datos IS
    SELECT codigo, stock
    FROM ARTICULO
    WHERE stock < 8
    ORDER BY stock;
  fila datos%ROWTYPE;
  mensaje CHAR(10);
BEGIN
  OPEN datos;
  LOOP
    FETCH datos INTO fila;
    EXIT WHEN datos%NOTFOUND;
    IF fila.stock >= 5 AND fila.stock <= 10 THEN
      mensaje := 'Normal';
    ELSIF fila.stock < 5 THEN
      mensaje := 'Bajo';
    ELSE
      mensaje := 'Exceso';
    END IF;
    DBMS_OUTPUT.PUT_LINE (fila.codigo || ' - ' || fila.stock || ' - ' || m
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('Total registros: ' || datos%ROWCOUNT);
  CLOSE datos;
```

```
END;

-- Solución versión 3 (bucle FOR)
SET SERVEROUTPUT ON
DECLARE
  CURSOR datos IS
    SELECT codigo, stock
    FROM ARTICULO
    WHERE stock < 8
    ORDER BY stock;
  mensaje CHAR(10);
BEGIN
  FOR fila IN datos LOOP
    IF fila.stock >= 5 AND fila.stock <= 10 THEN
      mensaje := 'Normal';
    ELSIF fila.stock < 5 THEN
      mensaje := 'Bajo';
    ELSE
      mensaje := 'Exceso';
    END IF;
    DBMS_OUTPUT.PUT_LINE (fila.codigo || ' - ' || fila.stock || ' - ' || m
  END LOOP;
END;
```

Cursores con Parámetros

```
CURSOR nombrecursor (parámetro1 tipodato, parámetro2 tipodato,...) IS instruc
```

Ejemplo 5: Cursor con Parámetros

Modificar el ejemplo 4 para seleccionar los artículos cuyo stock sea menor a un valor introducido como parámetro.

```
-- Solución (Ejemplo 4 versión 1)
SET SERVEROUTPUT ON
DECLARE
  CURSOR datos (minimo NUMBER) IS
    SELECT codigo, stock
    FROM ARTICULO
    WHERE stock < minimo
    ORDER BY stock;
  codigofila ARTICULO.CODIGO%TYPE;
  stockfila ARTICULO.STOCK%TYPE;
  mensaje CHAR(10);
  valor NUMBER;
BEGIN
  valor := 6;
  OPEN datos (valor);
  LOOP
    FETCH datos INTO codigofila, stockfila;
    EXIT WHEN datos%NOTFOUND;
    IF stockfila >= 5 AND stockfila <= 10 THEN
      mensaje := 'Normal';
    ELSIF stockfila < 5 THEN
      mensaje := 'Bajo';
    ELSE
      mensaje := 'Exceso';
    END IF;
    DBMS_OUTPUT.PUT_LINE (codigofila || ' - ' || stockfila || ' - ' || mer
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('Total registros: ' || datos%ROWCOUNT);
  CLOSE datos;
END;

-- Solución (Ejemplo 4 versión 3)
SET SERVEROUTPUT ON
DECLARE
  CURSOR datos (minimo NUMBER) IS
    SELECT codigo, stock
    FROM ARTICULO
    WHERE stock < minimo
    ORDER BY stock;
  mensaje CHAR(10);
  valor NUMBER;
BEGIN
  valor := 6;
  FOR fila IN datos (valor) LOOP
    IF fila.stock >= 5 AND fila.stock <= 10 THEN
      mensaje := 'Normal';
    ELSIF fila.stock < 5 THEN
      mensaje := 'Bajo';
    ELSE
      mensaje := 'Exceso';
    END IF;
    DBMS_OUTPUT.PUT_LINE (fila.codigo || ' - ' || fila.stock || ' - ' || m
  END LOOP;
END;
```

Excepciones

Para manejar los posibles errores que se producen al ejecutar un bloque, se puede dejar que el SGBD los gestione o controlarlos explícitamente utilizando la sección `EXCEPTION` de los bloques.

Sintaxis de la Sección EXCEPTION

```
EXCEPTION
  WHEN excepción1 THEN
    Instrucciones
  WHEN excepción2 THEN
    Instrucciones
  WHEN OTHERS THEN
    Instrucciones_a_realizar_en_otro_caso
END;
```

Tipos de Excepciones

- Internas de ORACLE (con nombre y sin nombre)
- Definidas por el usuario

Excepciones Internas con Nombre

Nombre	Número	Situación
CURSOR_ALREADY_OPEN	-06511	Se intenta abrir un cursor que ya está abierto.
DUP_VAL_ON_INDEX	-00001	Se intenta almacenar un valor duplicado en una columna que tiene un índice único.
INVALID_CURSOR	-01001	Se realiza una operación no válida sobre un cursor.
INVALID_NUMBER	-01722	Falla la conversión de una cadena de caracteres a un número.
LOGIN_DENIED	-01017	Se intenta conectar a ORACLE con un usuario o contraseña inválidos.
NO_DATA_FOUND	-01403	Una instrucción SELECT que devuelve como mucho una fila, no devuelve ninguna.
PROGRAM_ERROR	-06501	Problema interno.
ROWTYPE_MISMATCH	-06504	Tipos incompatibles de datos en una asignación o un parámetro.
STORAGE_ERROR	-06500	No hay memoria suficiente o está corrupta.
TIMEOUT_ON_RESOURCE	-00051	Se excede el tiempo máximo de espera para un recurso.
TOO_MANY_ROWS	-01422	Una instrucción SELECT que devuelve como mucho una fila, devuelve más de una.
VALUE_ERROR	-06502	Error aritmético, de conversión, truncamiento o redondeo, tamaño,...
ZERO_DIVIDE	-01476	División por cero.

Ejemplo 6

Mostrar el código y stock de los artículos que tienen en stock el número de unidades que el usuario introduce por teclado.

```
SET SERVEROUTPUT ON
DECLARE
  fila ARTICULO%ROWTYPE;
BEGIN
  SELECT *
  INTO fila
  FROM ARTICULO
  WHERE STOCK = &unidades;
  DBMS_OUTPUT.PUT_LINE ('El articulo es:');
  DBMS_OUTPUT.PUT_LINE (fila.codigo || ' - ' || fila.stock);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No hay articulos con ese stock');
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Hay mas de un articulo con ese stock');
END;
/
```

Excepciones Internas sin Nombre

1. Declarar una variable del tipo `EXCEPTION`.

   ```
   nombreexcepcion EXCEPTION;
   ```

2. Asociar esta variable al número de la excepción.

   ```
   PRAGMA EXCEPTION_INIT (nombreexcepcion, numeroexcepcion);
   ```

3. Incluir la excepción en la sección `EXCEPTION`.

Ejemplo 7

Actualizar el stock del artículo "AR11" sumándole el número de unidades que se introduce por teclado. Se ejecuta la excepción `stock_negativo` si el stock resultante es negativo.

```
SET SERVEROUTPUT ON
DECLARE
  stock_negativo EXCEPTION;
  PRAGMA EXCEPTION_INIT (stock_negativo, -02290);
  stock1 ARTICULO.STOCK%TYPE;
BEGIN
  UPDATE ARTICULO SET stock = stock + &unidades WHERE codigo = 'ART1';
  COMMIT;
  SELECT stock INTO stock1 FROM ARTICULO WHERE codigo = 'ART1';
  DBMS_OUTPUT.PUT_LINE ('El stock actualizado es: ' || stock1);
EXCEPTION
  WHEN stock_negativo THEN
    DBMS_OUTPUT.PUT_LINE ('No se realiza la actualizacion.');
    DBMS_OUTPUT.PUT_LINE ('El stock final no puede ser menor que 0.');
END;
/
```

Funciones SQLCODE y SQLERRM

- `SQLCODE`: Obtiene el número de error.
- `SQLERRM`: Obtiene el mensaje de error asociado al número.

Excepciones Definidas por el Programador

Opción 1:

1. Declarar una variable del tipo `EXCEPTION`.

   ```
   nombreexcepcion EXCEPTION;
   ```

2. Hacer "saltar" la excepción en el sitio correspondiente.

   ```
   RAISE nombreexcepcion;
   ```

3. Incluirla en el apartado `EXCEPTION`.

Opción 2:

Utilizar la función:

```
RAISE_APPLICATION_ERROR (numeroexcepcion, mensaje);
```

El número debe estar en el intervalo -20000 y -20999. Es preferible utilizar la opción 1.

Ejemplo 8

Añadir una excepción que se ejecute cuando no exista el artículo e imprima el mensaje "El artículo no existe.".

```
-- Opción 1
SET SERVEROUTPUT ON
DECLARE
  stock_negativo EXCEPTION;
  PRAGMA EXCEPTION_INIT (stock_negativo, -02290);
  stock1 ARTICULO.STOCK%TYPE;
  articulo_noexiste EXCEPTION;
BEGIN
  UPDATE ARTICULO SET stock = stock + &unidades WHERE codigo = 'ARTIC1';
  IF SQL%NOTFOUND THEN
    RAISE articulo_noexiste;
  END IF;
  COMMIT;
  SELECT stock INTO stock1 FROM ARTICULO WHERE codigo = 'ART1';
  DBMS_OUTPUT.PUT_LINE ('El stock actualizado es: ' || stock1);
EXCEPTION
  WHEN stock_negativo THEN
    DBMS_OUTPUT.PUT_LINE ('No se realiza la actualizacion.');
    DBMS_OUTPUT.PUT_LINE ('El stock final no puede ser menor que 0.');
  WHEN articulo_noexiste THEN
    DBMS_OUTPUT.PUT_LINE ('El articulo no existe.');
END;

-- Opción 2
SET SERVEROUTPUT ON
DECLARE
  stock_negativo EXCEPTION;
  PRAGMA EXCEPTION_INIT (stock_negativo, -02290);
  stock1 ARTICULO.STOCK%TYPE;
BEGIN
  UPDATE ARTICULO SET stock = stock + &unidades WHERE codigo = 'ARTIC1';
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR (-20100,'El articulo no existe.');
  END IF;
  COMMIT;
EXCEPTION
  WHEN stock_negativo THEN
    DBMS_OUTPUT.PUT_LINE ('No se realiza la actualizacion.');
    DBMS_OUTPUT.PUT_LINE ('El stock final no puede ser menor que 0.');
END;
```

Funciones

Las funciones realizan un cálculo y devuelven un valor. Se almacenan en la BD.

Sintaxis para Definir una Función

```
CREATE OR REPLACE FUNCTION nombrefuncion (parametros)
RETURN tipodatos
IS | AS
  Declaramos las variables
BEGIN
  Instrucciones del bloque
EXCEPTION
  Instrucciones para tratamiento de excepciones
END;
```

Si hay errores, consultar la vista `USER_ERRORS`.

```
SELECT * FROM USER_ERRORS WHERE NAME = 'NOMBREFUNCION';
```

Llamar a una Función

```
SELECT nombrefuncion FROM DUAL;
```

Ejemplo 9

La función `TOTAL_ARTICULOS` obtiene el número total de artículos en stock que hay en la tabla `ARTICULO`.

```
CREATE OR REPLACE FUNCTION TOTAL_ARTICULOS
RETURN NUMBER
IS
  total NUMBER := 0;
BEGIN
  SELECT SUM (stock) INTO total FROM ARTICULO;
  RETURN total;
END;
```

Ejemplo 10

La función recibe como parámetro el código de un artículo y devuelve el número de unidades en stock.

```
CREATE OR REPLACE FUNCTION STOCK_ARTICULO (codart ARTICULO.CODIGO%TYPE)
RETURN NUMBER
IS
  total NUMBER := 0;
BEGIN
  SELECT stock INTO total FROM ARTICULO WHERE codigo = codart;
  RETURN total;
END;
```

Eliminar una Función

```
DROP FUNCTION nombrefuncion;
```

Vista USER_PROCEDURES

Contiene información sobre los procedimientos y funciones creados por un usuario.

Procedimientos

Los procedimientos realizan una serie de acciones sobre la BD y se almacenan en la base de datos. Pueden tener parámetros de entrada (`IN`), salida (`OUT`) y de entrada y salida (`IN OUT`). Si no se especifica nada, el parámetro es de entrada.

Sintaxis

```
CREATE OR REPLACE PROCEDURE nombreprocedimiento (parametros)
IS | AS
  Declaramos las variables
BEGIN
  Instrucciones
EXCEPTION
  Instrucciones para tratamiento de excepciones
END;
```

Ejecutar un Procedimiento

```
EXECUTE nombreprocedimiento;
```

Eliminar un Procedimiento

```
DROP PROCEDURE nombreprocedimiento;
```

Ejemplo 11

El procedimiento recibe como argumento el código del artículo y el número de unidades y actualiza el stock de dicho artículo en la cantidad de unidades que se le indique.

```
CREATE OR REPLACE PROCEDURE REPONER_STOCK (codart ARTICULO.CODIGO%TYPE, ur
IS
  stock_negativo EXCEPTION;
  PRAGMA EXCEPTION_INIT (stock_negativo, -02290);
BEGIN
  UPDATE ARTICULO SET stock = stock + unidades WHERE codigo = codart;
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR (-20100,'El articulo no existe.');
  END IF;
  COMMIT;
EXCEPTION
  WHEN stock_negativo THEN
    DBMS_OUTPUT.PUT_LINE('El stock no puede ser menor que 0.');
END;
/
```

Ejecutar el Procedimiento

```
EXECUTE REPONER_STOCK('ART1',5);
```

Disparadores (Triggers)

Un disparador (trigger) es un código que se ejecuta automáticamente cuando se intenta realizar una instrucción sobre la BD. Se pueden diferenciar disparadores asociados a tablas.

Disparadores (Triggers)

Los disparadores son bloques de código PL/SQL que se ejecutan automáticamente en respuesta a ciertos eventos que ocurren en una base de datos. Generalmente, estos eventos son operaciones DML (INSERT, DELETE, UPDATE) sobre tablas o vistas. No consideraremos disparadores asociados al sistema, solo a tablas o vistas.

Aplicaciones de los Disparadores

Los disparadores son útiles para:

- Crear reglas de integridad complejas que no se pueden definir fácilmente con restricciones.
- Realizar cambios en la base de datos de manera transparente para el usuario.
- Validar datos y prevenir errores.

Sintaxis para crear un Disparador

La sintaxis general para crear un disparador es la siguiente:

```
CREATE OR REPLACE TRIGGER nombretrigger
(BEFORE | AFTER | INSTEAD OF)
(INSERT | DELETE | UPDATE [OF columnas] [OR INSERT | DELETE | UPDATE [OF c
ON nombretabla
[FOR EACH ROW [WHEN (condicion)]]
DECLARE
  -- Declaración de variables
BEGIN
  -- Instrucciones
  [EXCEPTION
    -- Instrucciones para tratamiento de excepciones]
END;
```

Componentes de un Disparador

1. Evento:
    - Define qué comando DML (INSERT, DELETE, UPDATE) activa el disparador y a qué tabla está asociado.
    - Especifica cuándo se ejecuta el disparador en relación con el comando DML:
        - BEFORE: Antes de ejecutar el comando DML.
        - AFTER: Después de ejecutar el comando DML.
        - INSTEAD OF: Sustituye el comando DML con las instrucciones del disparador (útil para vistas que no permiten instrucciones DML).
2. Tipo de Disparador:
    - Disparador tipo fila (FOR EACH ROW): Las instrucciones del disparador se ejecutan para cada fila afectada por el evento. Se puede agregar una condición `WHEN` para que las instrucciones solo se ejecuten si la fila cumple con esa condición.
    - Disparador tipo instrucción: Las instrucciones del disparador se ejecutan una sola vez cuando se produce el evento que lo activa.
3. Acción:
    - Es la parte del disparador que contiene las acciones a realizar.
    - Se implementa en código PL/SQL dentro de las secciones `DECLARE` y `BEGIN...END`.

Referenciando Valores Antiguos y Nuevos

Dentro del código PL/SQL del disparador, se pueden referenciar los valores antiguos y nuevos de las columnas de una fila utilizando `NEW.nombrecolumna` y `OLD.nombrecolumna`, respectivamente.

Nota: En la condición `WHEN` de `FOR EACH ROW`, no se usa `:` delante de `NEW` y `OLD`.

La siguiente tabla muestra cuándo están disponibles las variables `:NEW` y `:OLD` para referenciar los valores nuevos y antiguos de una columna:

Evento	NEW	OLD
INSERT	Sí	NULL
DELETE	NULL	Sí
UPDATE	Sí	Sí

Múltiples Eventos y Comandos Condicionales

Un disparador puede activarse debido a varios eventos (usando `OR`). Para diferenciar qué evento activó el disparador dentro de la sección `BEGIN...END`, se utilizan los comandos:

- `INSERTING`: Devuelve verdadero si el disparador se activó por una instrucción `INSERT`.
- `DELETING`: Devuelve verdadero si el disparador se activó por una instrucción `DELETE`.
- `UPDATING`: Devuelve verdadero si el disparador se activó por una instrucción `UPDATE`.
- `UPDATING(nombrecolumna)`: Devuelve verdadero si el disparador se activó por una instrucción `UPDATE` que actualizó la columna `nombrecolumna`.

Estos comandos se utilizan generalmente junto con una instrucción condicional (`IF`).

Operaciones de Mantenimiento

- Activar/Desactivar:
  - Por defecto, un disparador se crea en estado `ENABLE` (activado).
  - Para desactivar un disparador y evitar que se ejecute, se debe cambiar su estado a `DISABLE` (desactivado).
  - Para activar/desactivar un disparador individual:

```
ALTER TRIGGER nombretrigger (ENABLE | DISABLE);
```

  - Para activar/desactivar todos los disparadores de una tabla:

```
ALTER TABLE nombretabla (ENABLE | DISABLE) ALL TRIGGERS;
```

- Eliminar:
  - Para eliminar un disparador:

```
DROP TRIGGER nombretrigger;
```

- Consultar:
  - La información sobre los disparadores creados por un usuario se encuentra en la vista `USER_TRIGGERS`.
  - Se puede usar `DESCRIBE` para ver la estructura de la vista y luego realizar consultas (`SELECT`) sobre los campos deseados.

Ejemplos Prácticos

Ejemplo 12: Registrar Modificaciones en la Tabla ARTICULO

Este disparador se ejecuta cada vez que se inserta o actualiza un registro en la tabla `ARTICULO` y almacena información sobre el cambio en la tabla `ARTICULO_MOVIMIENTO`.

Estructura de las tablas:

- `ARTICULO (codigo VARCHAR (6), stock NUMBER (3,0)) -- CP: codigo`
- `ARTICULO_MOVIMIENTO (codigo VARCHAR (6), stock NUMBER (3,0), fecha DATE, accion VARCHAR (1), usuario VARCHAR(15))`

```
CREATE OR REPLACE TRIGGER MODIFICAR_ARTICULO
BEFORE INSERT OR UPDATE ON ARTICULO
FOR EACH ROW
DECLARE
  tipo_movimiento VARCHAR (1);
BEGIN
  IF INSERTING THEN
    tipo_movimiento := 'I';
  ELSE
    tipo_movimiento := 'M';
  END IF;
  INSERT INTO ARTICULO_MOVIMIENTO VALUES (:NEW.codigo, :NEW.stock, SYSDATE
END;
/
```

Ejemplo 13: Control de Stock al Insertar un Pedido

Este disparador se ejecuta antes de insertar un nuevo pedido en la tabla `PEDIDO`. Verifica si hay suficiente stock disponible en la tabla `ARTICULO`. Si no hay suficiente stock, se crea un registro en la tabla `SUMINISTRAR` indicando las unidades pendientes de suministro.

Estructura de las tablas:

- `ARTICULO (codigo VARCHAR (6), stock NUMBER (3,0)) -- CP: codigo`
- `PEDIDO (numero NUMBER, codart VARCHAR (6), unidades NUMBER (3,0), estado VARCHAR (15)) -- CP: numero, Cajena: codart -> ARTICULO`
- `SUMINISTRAR (numpedido NUMBER, codart VARCHAR(6), unidades NUMBER (3,0), usuario VARCHAR (15), fecha DATE)`

```
CREATE OR REPLACE TRIGGER PEDIR_ARTICULO
BEFORE INSERT ON PEDIDO
FOR EACH ROW
DECLARE
  stockactual NUMBER;
  stockfaltante NUMBER;
BEGIN
  SELECT stock INTO stockactual FROM ARTICULO WHERE ARTICULO.codigo = :NEW
  IF stockactual >= :NEW.unidades THEN
    UPDATE ARTICULO SET stock = stockactual - :NEW.unidades WHERE codigo =
  ELSE
    UPDATE ARTICULO SET stock = 0 WHERE codigo = :NEW.codart;
    stockfaltante := :NEW.unidades - stockactual;
    :NEW.estado := 'Pendiente stock ' || stockfaltante;
    INSERT INTO SUMINISTRAR VALUES (:NEW.numero, :NEW.codart, stockfaltant
  END IF;
END;
/
```

Ejemplo 14: Restricción de Eliminación de Departamento

Este disparador se ejecuta antes de eliminar un departamento de la tabla `DEPARTAMENTO`. Solo permite la eliminación si el departamento tiene menos de tres empleados. En caso contrario, lanza un error.

Estructura de las tablas:

- `DEPARTAMENTO (codigo VARCHAR (10), nombre VARCHAR(20)) -- CP: codigo`
- `EMPLEADO (dni VARCHAR (10), nombre VARCHAR (20), apellidol VARCHAR(20), depto VARCHAR (10)) -- CP: dni, Cajena: depto -> DEPARTAMENTO`

```
SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER ELIMINAR_DEPTO
BEFORE DELETE ON DEPARTAMENTO
FOR EACH ROW
DECLARE
  totalempleados NUMBER;
BEGIN
  SELECT COUNT(*) INTO totalempleados FROM EMPLEADO WHERE EMPLEADO.depto =
  IF totalempleados > 2 THEN
    RAISE_APPLICATION_ERROR(-20001, 'El departamento ' || :OLD.codigo || '
  ELSE
    DELETE FROM EMPLEADO WHERE EMPLEADO.depto = :OLD.codigo;
  END IF;
END;
/
```<script type="text/javascript">
  window.MathJax = {
    tex: {
      inlineMath: [['$', '$'], ['\\(', '\\)']],
      displayMath: [['$$', '$$'], ['\\[', '\\]']]
    }
  };
</script>
<script type="text/javascript" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-chtml.js">
</script>